

Name: Jadhav Laxman Udhav

Roll No.: SY_A_ 138

Subject: DAA Practical

Experiment 1

Aim: Program to implement linear, binary search using recursion Program:

```
#include<stdio.h>
```

```
#include<conio.h>
```

```
int binary_recursive(int a[], int low, int high, int x); int
```

```
binary_iterative(int a1[], int low1, int high1, int x1);
```

```
int main(){
```

```
    int
```

```
    ch;
```

```
    int num, arr[100], i, key;    int lower_bound,
```

```
    upper_bound, result_recursive;
```

```
    int res_iterative,flag;
```

```
    while(num > 0)
```

```
    {        printf("\n-----
```

```
---");
```

```
printf("\n 1.Iterative Binary \n 2.Recursive Binary \n 3.Linear Search");        printf("\n---
```

```
-----");        printf("\nEnter Your
```

```
Choice:");        scanf("%d",&ch);
```

```
        switch(ch)
```

```

        {           case
1:
printf("\nEnter
Total Number of
Elements:");
scanf("%d",&num
);

        printf("\nEnter %d Elements:",num);
for(i=0; i<num; i++)
    {
scanf("%d",&arr[i]);
    }

        printf("\n Enter Element You want To Search:");
scanf("%d",&key);
        lower_bound
=0;
upper_bound = num-1;

        res_iterartive = binary_iterative(arr,lower_bound,upper_bound,key);

        if(res_iterartive == -1)
        {
            printf("\nElement Not Found using Bimary Iteration:");
        }           else
        {
            printf("\nElement is Found Using Binary Iteration");

```

```
printf("At location %d",res_iterartive + 1);
```

```
}
```

```
break;
```

```
case 2:
```

```
printf("\nEnter
```

```
Total Number
```

```
of Elements:");
```

```
scanf("%d",&n
```

```
um);
```

```
printf("\n Enter %d Elements:",num);
```

```
for(i=0; i<num; i++)
```

```
{
```

```
scanf("%d",&arr[i]);
```

```
}
```

```
printf("\n Enter Element You want To Search:");
```

```
scanf("%d",&key);
```

```
lower_bound
```

```
=0;
```

```
upper_bound =num-1;
```

```
result_recursive = binary_recursive(arr,lower_bound,upper_bound,key);
```

```
if(result_recursive == -1)
```

```
{
```

```
printf("\nElement is Not Found using Binary Recursive:");
```

```
        }  
    else {  
        printf("\nElement  
is Found using  
Binary  
Recursive");  
        printf("\n At  
Location %d",  
result_recursive  
e+1);  
    }
```

```
break;
```

```
case 3:
```

```
    printf("\nEnter Total Number of Elements:");    scanf("%d",&num);
```

```
    printf("\nEnter %d Elements:",num);    for(i=0;  
i<num; i++)
```

```
    {  
scanf("%d",&arr[i]);  
    }
```

```
    printf("\nEnter Element You want To Search:");    scanf("%d",&key);
```

```
    for(i=0; i<num; i++)  
    {  
if(arr[i] == key)
```

```

        {
            flag =1;
break;
        }

    }

    if(flag ==
1)

    {
        printf("\nElement is Found using Linear Search");
printf("\n At Location %d",i+1);
    }        else
    {
        printf("\nElement is Not Founf Using Linear Search");
    }

break;
default:
    printf("Error");
    }
return
0;

    }
}

```

```

int binary_iterative(int a[], int low, int high, int x)

```

```

{   int mid1;
if(low
>high)
return -1;
while(low
<= high)

{
    mid1 = (low + high) /2;

    if(x == a[mid1])
return mid1;    else if(x <
a[mid1])        return high =
a[mid1-1];    else
return low = a[mid1+1];
    return -
1;

}
}

```

```

int binary_recursive(int a1[], int low1, int high1, int x1)
{   int mid;

    if(low1 > high1)    return
-1;

    mid =(low1 + high1) /2;

```

```

        if(x1 == a1[mid])        return
mid;

        else if(x1 < a1[mid])    return
binary_recursive(a1,low1,mid-1,x1);    else
        return binary_recursive(a1,mid+1,high1,x1);

}

```

Output:

The screenshot shows a Visual Studio Code editor with a C program named 'Searching.c' open. The program implements a binary search algorithm. The output window shows the following interaction:

```

-----
1.Iterative Binary
2.Recursive Binary
3.Linear Search
-----
Enter Your Choice:1

Enter Total Number of Elements:4

Enter 4 Elements:11
45
67
89

Enter Element You want To Search:45

Element is Found Using Binary IterationAt location 2
-----
1.Iterative Binary
2.Recursive Binary
3.Linear Search
-----
Enter Your Choice:

```

At the bottom of the terminal, an error message is visible: `c:/mingw/bin/../lib/gcc/mingw32/6.3.0/../../../../mingw32/bin/ld.exe: cannot open output file Searching.exe: Permission denied collect2.exe: error: ld returned 1 exit status`. The Windows taskbar at the bottom shows the date as 22-05-2021 and the time as 22:35.

Q. 1. Analyze Binary Search. Find time complexity of Binary search & Linear search for best case & worst case. Write posteriori analysis of program for binary search(for both Best case & Worst case).

Ans: The time complexity of the binary search algorithm is $O(\log n)$. The best-case time complexity would be $O(1)$. And the worst case complexity is $O(\log n)$.

In linear search, best-case complexity is $O(1)$ where the element is found at the first index. ... The worst-case complexity is $O(\log 2n)$.

Experiment 2

Aim: Program to find minimum / maximum, kth smallest element using D & C.

Program:

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <conio.h>
```

```
#include <time.h>
```

```
int max, min; int
```

```
a[50];
```

```
void maxmin(int i , int j)
```

```
{    int max1, min1,
```

```
mid;    if(i == j)    {
```

```
max = min = a[i];
```

```
    }    else
```

```
{
```

```
    if( i == j-1)
```

```
    {
```

```
if(a[i] < a[j])
```

```
    {
```

```
max = a[j];
```

```
min = a[i];
```

```
    }
```



```

        else {
max = a[i];      min
= a[j];
        } }
else {      mid =
(i + j ) /2;
maxmin(i, mid);
max1 = max;
min1 = min;

        maxmin(mid+1, j);
        if(max < max1)
max = max1;      else
if(min > min1)
min = min1;
        }

    }
}

```

```

int main() {    int i,
num;    clock_t st,
end;    double
totaltime;

```

```

    st = clock();

```

```
printf("Enter total number of elements:\n"); scanf("%d",
&num);
```

```
printf("Enter the elements of array: \n"); for(i=1;
i<=num; i++)
```

```
{
    scanf("%d",&a[i]);
}
```

```
max = a[0]; min
= a[0];
```

```
maxmin(1, num);
```

```
printf("Minimum element: %d : \n", min); printf("Maximum
element: %d : \n", max);
```

```
end = clock();
```

```
totaltime = ((double) (end - st)) / CLOCKS_PER_SEC; printf("Total
time : %f",totaltime);
```

```
getch();
}
```

Output:

Experiment 3

Aim: Program for Quick sort using Divide and Conquer

Program:

```
#include<stdio.h>
#include <stdlib.h>
#include<conio.h>
#include <time.h>

void quicksort(int number[25],int first,int last)
{   int i, j, pivot,
temp;
if(first<last)   {
pivot=first;     i=first;
j=last;

while(i<j)
{
while(number[i]<=number[pivot]&& i<last)
{   i++;
}

while(number[j]>number[pivot])
{   j--;
}   if(i<j)
{
temp=number[i];    number[i]=number[j];
number[j]=temp;
```

```

    }
}
temp=number[pivot];
number[pivot]=number[j];    number[j]=temp;
quicksort(number,first,j-1);
quicksort(number,j+1,last); } } int main(){    int
i, total, number[25];    clock_t st, end;    double
totaltime;

st = clock();

printf("Enter total no of elements: ");    scanf("%d",&total);

printf("Enter %d elements: ", total);

for(i=0;i<total;i++)
{
    scanf("%d",&number[i]);
}

quicksort(number,0,total-1);    printf("Sorted
Elements are: ");    for(i=0;i<total;i++)
{
    printf(" %d",number[i]);
}

end = clock();

```

```

    totaltime = ((double) (end - st)) / CLOCKS_PER_SEC;    printf("\nTotal
time : %f",totaltime);

getch();
return 0;
}

```

Output:

The screenshot shows a Visual Studio Code editor with a C++ file named 'QuickSort.cpp'. The code is as follows:

```

DAA Prac > C:\QuickSort\ > QuickSort.cpp [25, 1st, #0]
1 #include<stdio.h>
2 #include<stdlib.h>

```

The console output shows the program's execution:

```

D:\DAA Prac\QuickSort.exe
Enter total no. of elements: 6
Enter 6 elements: 263
271
32
44
256
24
33
Sorted Elements are: 32 33 44 256 271 456
Total time : 7.731000

```

The bottom status bar indicates the program is running with the command: `cd "C:\DAA Prac\" && gcc QuickSort.c -o QuickSort -BA "D:\DAA Prac\QuickSort"`. The Windows taskbar at the bottom shows the date and time as 23-05-2021, 15:06.

Q.3. Generate Recurrence relation for Quick sort & solve the same. Write posteriori analysis of Quick sort program Worst Case Analysis Recurrence Relation:

$T(0) = T(1) = 0$ (base case)

$T(N) = N + T(N-1)$ Solving

the RR:

$T(N) = N + T(N-1)$

$T(N-1) = (N-1) + T(N-2)$ $T(N-2)$

$$= (N-2) + T(N-3)$$

...

$$T(3) = 3 + T(2)$$

$$T(2) = 2 + T(1)$$

$$T(1) = 0$$

Hence,

$$T(N) = N + (N-1) + (N-2) \dots + 3 + 2$$

$$\approx N^2$$

$$2$$

QUICKSORT

Best Case Analysis

Recurrence Relation:

$$T(0) = T(1) = 0 \quad (\text{base case})$$

$$T(N) = 2T(N/2) + N$$

Solving the RR:

$$\frac{T(N)}{N} = \frac{N}{N} + \frac{2T(N/2)}{N}$$

Note: Divide both side of recurrence relation by N

$$\frac{T(N)}{N} = 1 + \frac{T(N/2)}{N/2}$$

$$\frac{T(N/2)}{N/2} = 1 + \frac{T(N/4)}{N/4}$$

$$\frac{T(N/4)}{N/4} = 1 + \frac{T(N/8)}{N/8}$$

...

$$\frac{T(\frac{N}{N/2})}{\frac{N}{N/2}} = 1 + \frac{T(\frac{N}{N})}{\frac{N}{N}} = 1 + \frac{T(1)}{1}$$

same as

$$\frac{T(2)}{2} = 1 + \frac{T(1)}{1}$$

Note: $T(1) = 0$

Hence,

$$\frac{T(N)}{N} = 1 + 1 + 1 + \dots 1$$

Note: $\log(N)$ terms

$$\frac{T(N)}{N} = \log N$$

$$T(N) = N \log N \quad \text{which is } O(N \log N)$$

Experiment 4

Aim: Program for Merge sort using Divide and Conquer.

Program:

```
#include<stdio.h>
```

```
#include<conio.h>
```

```
#include<time.h>
```

```
void merge(int a[], int l, int mid, int h)
```

```
{    int temp[50];
```

```
    int
```

```
    i ,j, k;
```

```
        k = 0;    i
```

```
= l;    j = mid
```

```
+
```

```
1;;
```

```
    while (i<= mid && j <= h)
```

```
    {        if(a[i] <
```

```
a[j])        {
```

```
            temp[k++] = a[i++];
```

```
        }        else
```

```
        {
```

```
            temp[k++] = a[j++];
```

```
        }
```

```
    }
```

```
    while (i <= mid)
```

```
    {
```

```
    temp[k++] = a[i++];  
}
```

```
while (j <= h)  
{  
    temp[k++] = a[j++];  
}
```

```
for(i = h; i >= l; i-- )  
{    a[i] = temp[-k];  
}
```

```
}
```

```
void mergesort(int a[],int l, int h)  
{    int mid;  
    if(l < h)    {        mid = (l  
+h) /2;        mergesort(a, l,  
mid);        mergesort(a , mid+1,  
h);        merge(a, l, mid, h);  
    }  
}
```

```
void disp(int a[], int size)
```

```
{  
int i;  
  
    for(i=0; i<size; i++)  
    {  
printf("%d\n",a[i]);  
        //printf("\n");  
    }  
printf("\n");  
}
```

```
int main() {    int  
arr[50], num, i;  
clock_t st, end;    double  
totaltime;
```

```
    st = clock();
```

```
    printf("Enter Total Number Of Elements:\n");    scanf("%d",  
&num);
```

```
    printf("Enter %d Elements: \n", num);    for(i=0;  
i<num; i++)  
    {  
        scanf("%d",&arr[i]);  
    }
```

```
    //size =sizeof(arr) / sizeof(arr[0]);
```

```

mergesort(arr, 0, num-1);

printf("Sorted Array :\n");   disp(arr,
num);

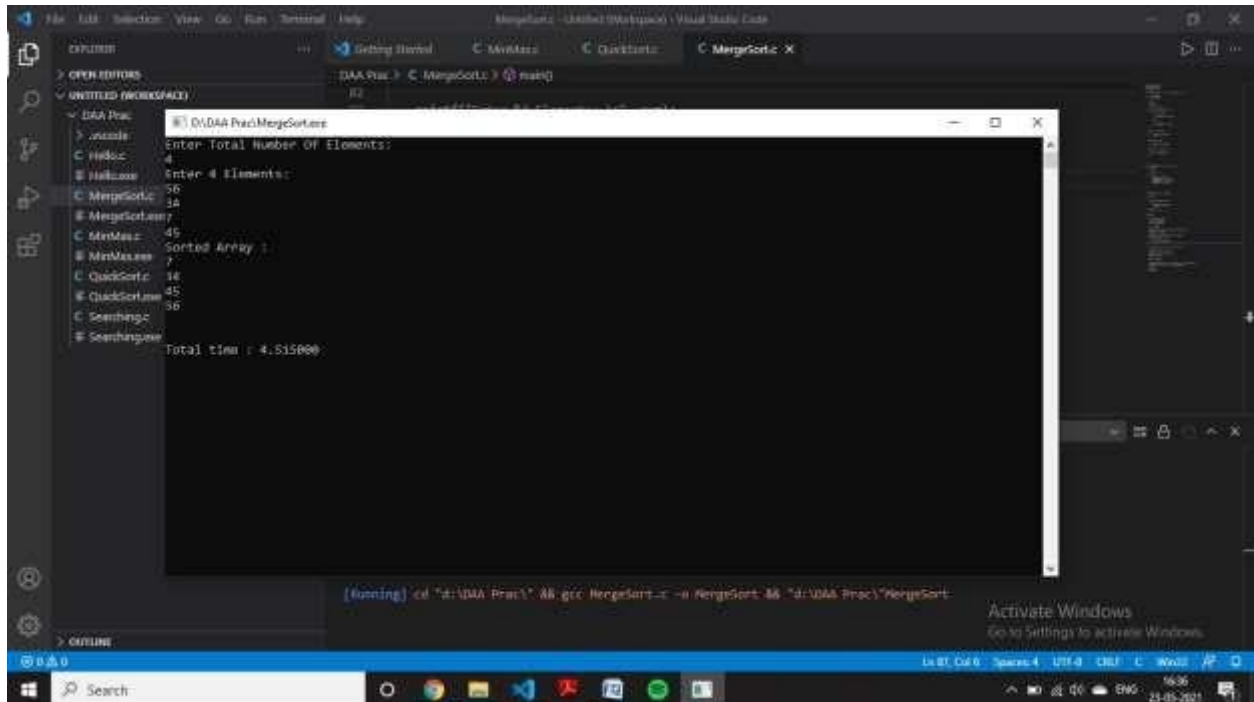
end = clock();

totaltime = ((double) (end - st)) / CLOCKS_PER_SEC;   printf("\nTotal
time : %f",totaltime);

getch();
return 0;
}

```

Output:



The screenshot shows a Visual Studio Code editor with a C program running. The console output is as follows:

```

D:\DAA Prac\MergeSort.exe
Enter Total Number Of Elements:
4
Enter 4 elements:
56
34
45
7
Sorted Array :
4
5
7
34
56
Total time : 4.515000

```

The taskbar at the bottom shows the system clock as 16:36 on 23-09-2021.

Q. 4. Why is it necessary to have auxiliary array in Merge function? Comment on space complexity of Merge sort. Write posteriori analysis of Merge sort program

Sorting arrays on different machines. Merge Sort is a recursive algorithm and time complexity can be expressed as following recurrence relation.

$$T(n) = 2T(n/2) + \theta(n)$$

The above recurrence can be solved either using the Recurrence Tree method or the Master method. It falls in case II of Master Method and the solution of the recurrence is $\theta(n \log n)$. Time complexity of Merge Sort is $\theta(n \log n)$ in all 3 cases (worst, average and best) as merge sort always divides the array into two halves and takes linear time to merge two halves. Auxiliary Space: $O(n)$

Experiment 5

Aim: Program to implement Fractional Knapsack problem using Greedy method Program:

```
#include<stdio.h>
#include<conio.h>
#include<time.h>

int main() {
    float weight[50],profit[50],ratio[50],Totalvalue,temp,capacity,amount;    int
    n,i,j;

    clock_t st, end;
    double totaltime;

    st = clock();

    printf("Enter the number of items :");
    scanf("%d",&n);    for (i = 0; i < n; i++)
    {
        printf("Enter Weight and Profit for item[%d] :\n",i);
        scanf("%f %f", &weight[i], &profit[i]);
    }
    printf("Enter the capacity of knapsack :\n");
    scanf("%f",&capacity);

    for(i=0;i<n;i++)
    ratio[i]=profit[i]/weight[i];    for (i = 0; i < n;
    i++)
```

```

for (j = i + 1; j < n; j++)
if (ratio[i] < ratio[j])
{
    temp = ratio[j];
ratio[j] = ratio[i];
ratio[i] = temp;

    temp = weight[j];
weight[j] = weight[i];    weight[i]
= temp;

    temp =
profit[j];
profit[j] = profit[i];    profit[i]
= temp;
}

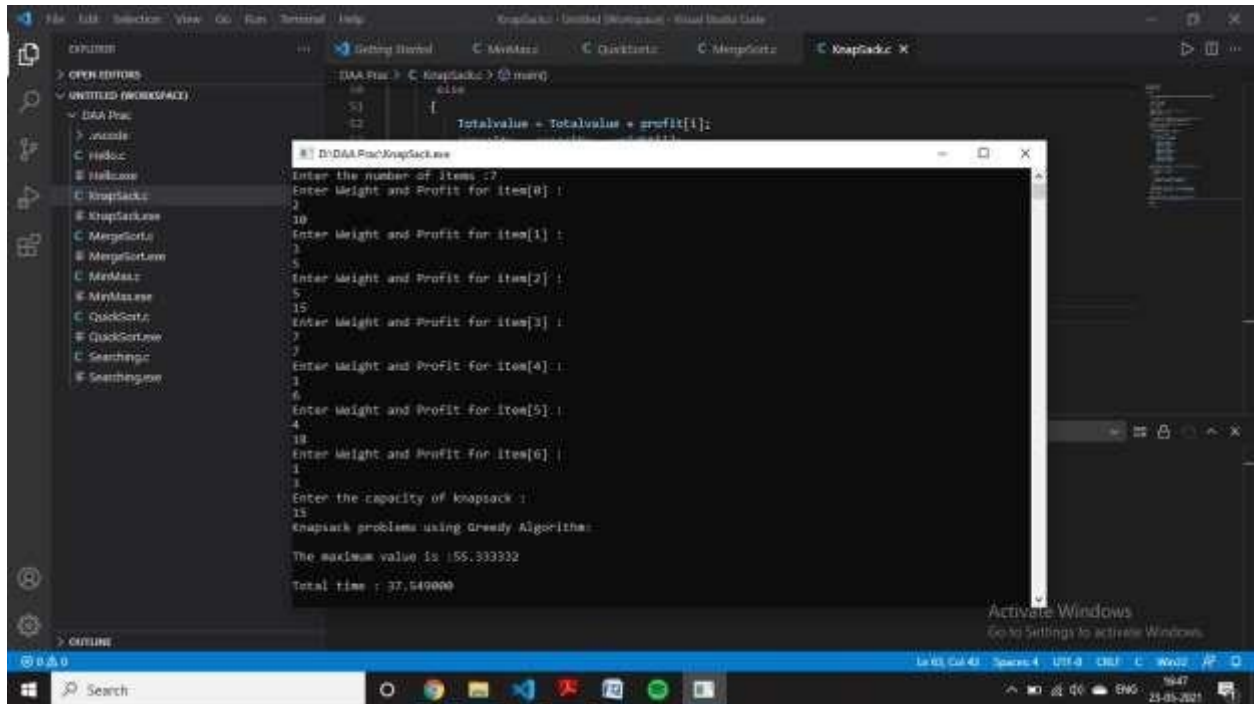
printf("Knapsack problems using Greedy Algorithm:\n");
for (i = 0; i < n; i++)
{
    if (weight[i] > capacity)
break;    else
{
    Totalvalue = Totalvalue + profit[i];    capacity
= capacity - weight[i];

    }    }    if
(i < n)
Totalvalue =
Totalvalue +
(ratio[i]*capacity

```

```
);  
printf("\nThe  
maximum value  
is  
:%f\n",Totalvalu  
e);  
  
end = clock();  
  
totaltime = ((double) (end - st)) / CLOCKS_PER_SEC;  printf("\nTotal  
time : %f",totaltime);  
  
getch();  
  
return 0;  
}
```

Output:



Q. 5. Differentiate between divide & conquer method and greedy technique. Write posteriori analysis for same

Divide & Conquer Method	Dynamic Programming
<p>1. It deals (involves) three steps at each level of recursion: Divide the problem into a number of subproblems. Conquer the subproblems by solving them recursively. Combine the solution to the subproblems into the solution for original subproblems.</p>	<p>1. It involves the sequence of four steps:</p> <ul style="list-style-type: none"> Characterize the structure of optimal solutions. Recursively defines the values of optimal solutions. Compute the value of optimal solutions in a Bottom-up minimum. Construct an Optimal Solution from computed information.
2. It is Recursive.	2. It is non Recursive.
3. It does more work on subproblems and hence has more time consumption.	3. It solves subproblems only once and then stores in the table.
4. It is a top-down approach.	4. It is a Bottom-up approach.
5. In this subproblems are independent of each other.	5. In this subproblems are interdependent.
6. For example: Merge Sort & Binary Search etc.	6. For example: Matrix Multiplication.

Experiment 6

Aim: Program to implement Prim's Algorithm to find minimum cost spanning tree using greedy method.

Program:

```
#include<stdio.h>
#include<conio.h>
{
int n, cost[10][10];

void prim() {   int i, j, startVertex, endVertex;
int k, nr[10], temp, minimumCost = 0, tree[10][3];

    /* For first smallest edge */
temp = cost[0][0];   for (i =
0; i < n; i++) {     for (j = 0; j
< n; j++) {         if (temp >
cost[i][j]) {         temp =
cost[i][j];         startVertex
= i;         endVertex = j;
        }
    }
}

    /* Now we have first smallest edge in graph */
tree[0][0] = startVertex;   tree[0][1] =
endVertex;   tree[0][2] = temp;   minimumCost
= temp;
```

```

/* Now we have to find min dis of each vertex from either startVertex
or endVertex by initialising nr[] array
*/

for (i = 0; i < n; i++) {    if
(cost[i][startVertex] < cost[i][endVertex])
nr[i] = startVertex;    else    nr[i] = endVertex;
}

/* To indicate visited vertex initialise nr[] for them to 100 */    nr[startVertex]
= 100;    nr[endVertex] = 100;

/* Now find out remaining n-2 edges */
temp = 99;    for (i = 1; i < n - 1; i++) {    for
(j = 0; j < n; j++) {
    if (nr[j] != 100 && cost[j][nr[j]] < temp) {
temp = cost[j][nr[j]];        k = j;
    }
}

/* Now i have got next vertex */    tree[i][0]
= k;    tree[i][1] = nr[k];    tree[i][2] =
cost[k][nr[k]];    minimumCost = minimumCost
+ cost[k][nr[k]];    nr[k] = 100;

/* Now find if k is nearest to any vertex    than
its previous near value */

```

```

        for (j = 0; j < n; j++) {            if (nr[j] != 100
&& cost[j][nr[j]] > cost[j][k])            nr[j] = k;
        }
        temp = 99;
    }

    /* Now i have the answer, just going to print it */
    printf("\nThe min spanning tree is:- "); for (i = 0; i
< n - 1; i++) { for (j = 0; j < 3; j++)
    printf("%d", tree[i][j]); printf("\n");
    }

    printf("\nMin cost : %d", minimumCost);
}

void main() { int
i, j;

    //clrscr();

    printf("\nEnter the no. of vertices :");
    scanf("%d", &n);

    printf("\nEnter the costs of edges in matrix form :"); for
(i = 0; i < n; i++) for (j = 0; j < n; j++) {
    scanf("%d", &cost[i][j]);
    }

    printf("\nThe matrix is : "); for
(i = 0; i < n; i++) { for (j = 0; j
< n; j++) { printf("%d\t",
cost[i][j]);

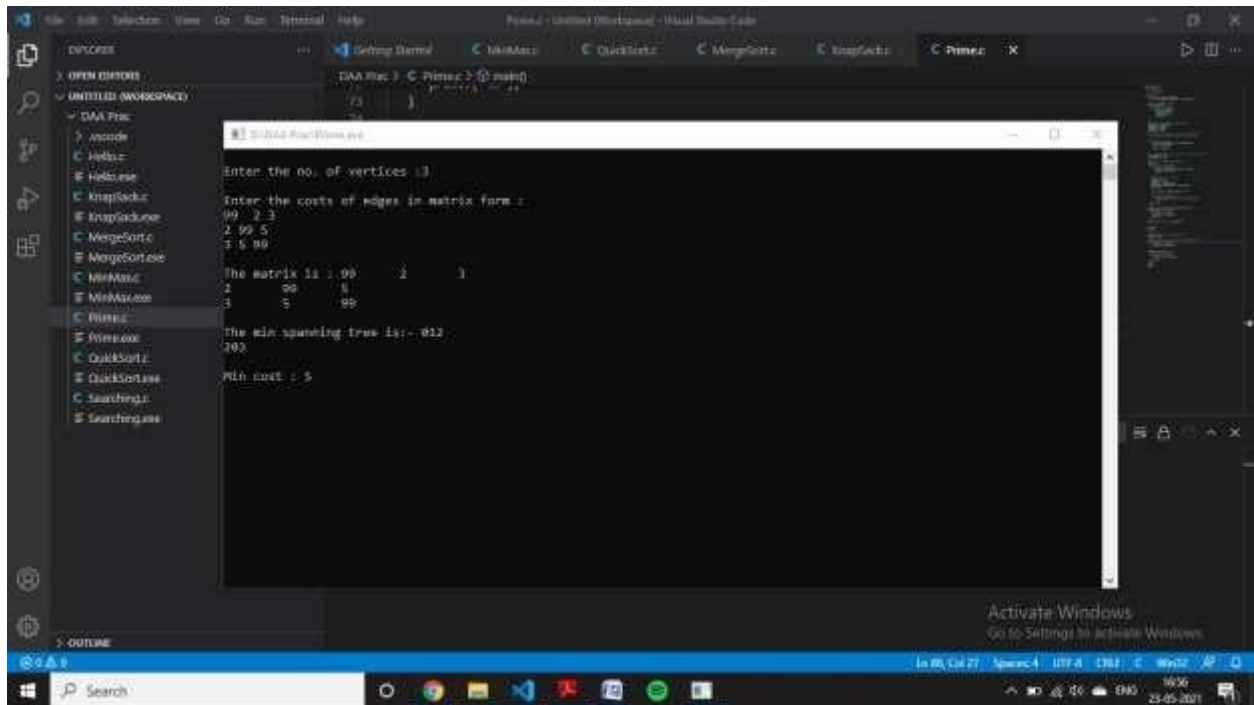
```

```

    }
printf("\n"); }
prim();
getch(); }

```

Output:



Q. 6. Define minimum spanning tree with example. Also write posteriori analysis for same.

Solve any one example using Prim's and Kruskal's algorithm

A minimum spanning tree is a special kind of tree that minimizes the lengths (or “weights”) of the edges of the tree. An example is a cable company wanting to lay line to multiple neighborhoods; by minimizing the amount of cable laid, the cable company will save money.

A tree has one path joins any two vertices. A spanning tree of a graph is a tree that:

- Contains all the original graph's vertices. Reaches out to (spans) all vertices.
- Is acyclic. In other words, the graph doesn't have any nodes which loop back to itself.

Experiment No. 7

Aim: Program for Topological Ordering of Vertices in given a digraph.

Program:

```
#include<stdio.h>
```

```
#include<conio.h>
```

```
int main(){
```

```
int i,j,k,n,a[10][10],indeg[10],flag[10],count=0;
```

```
printf("Enter the no of vertices:\n"); scanf("%d",&n); printf("Enter  
the adjacency matrix:\n");
```

```
for(i=0;i<n;i++){
```

```
    printf("Enter row %d\n",i+1);
```

```
    for(j=0;j<n;j++){
```

```
        scanf("%d",&a[i][j]);
```

```
    }
```

```
for(i=0;i<n;i++){
```

```
    indeg[i]=0;
```

```
    flag[i]=0;
```

```

    }

    for(i=0;i<n;i++)

        for(j=0;j<n;j++)

            indeg[i]=indeg[i]+a[j][i];           printf("\nThe
topological order is:");

    while(count<n){

        for(k=0;k<n;k++){

            if((indeg[k]==0) && (flag[k]==0)){

                printf("%d ",(k+1));

flag
[k]=1;

            }

            for(i=0;i<n;i++){
if(a[i][k]==1) indeg[k]--; } }

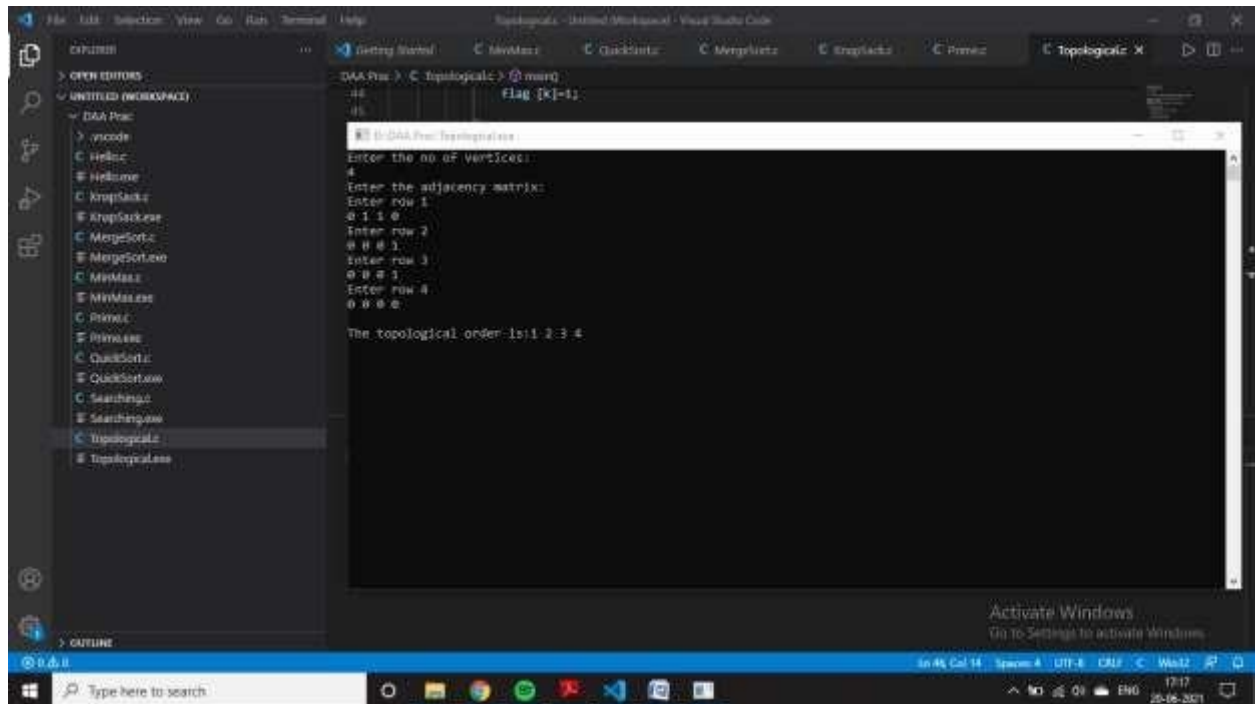
count++;

        }           getch();
return 0;

    }

```

Output:



The screenshot shows a Visual Studio Code window with a terminal running a C++ program for topological sorting. The program prompts the user to enter the number of vertices and an adjacency matrix. The user input is as follows:

```
Enter the no. of vertices: 4
Enter the adjacency matrix:
Enter row 1: 0 1 1 0
Enter row 2: 0 0 0 1
Enter row 3: 0 0 0 1
Enter row 4: 0 0 0 0
```

The program then outputs the topological order: 1 2 3 4.

```
The topological order is: 1 2 3 4
```

The Visual Studio Code interface shows the file explorer on the left with a list of files including 'Topological.cpp'. The terminal window is titled 'DAA-Prac > C++ Topological > Run'.

Experiment No: 8

Aim: Program to implement Warshall's algorithm/ Floyd warshall's algorithm for solving all pairs Shortest Path problem.

Program:

```
#include<stdio.h>
```

```
#include<conio.h>
```

```
int i, j, k,n,dist[10][10];
```

```
void floydWarshell ()
```

```
{   for (k = 0; k < n; k++)   for (i = 0;
```

```
i < n; i++)   for (j = 0; j < n; j++)
```

```
if (dist[i][k] + dist[k][j] < dist[i][j])
```

```
dist[i][j] = dist[i][k] + dist[k][j];
```

```
}
```

```
int main() {
```

```
    int i,j;
```

```
    printf("enter no of vertices :");   scanf("%d",&n);
```

```
    printf("\n");
```

```
    for(i=0;i<n;i++)   for(j=0;j<n;j++)
```

```
{       printf("dist[%d][%d]:",i,j);
```

```
scanf("%d",&dist[i][j]);
```

```
}
```

```

floydWarshell();

printf (" \n\n shortest distances between every pair of vertices \n");

for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
printf ("%d\t", dist[i][j]);    printf("\n");
} getch();

return 0;
}

```

Output:

The screenshot shows a Windows IDE with a project named 'Floyd's Algo'. The output window displays the following text:

```

enter no of vertices : 4
dist[0][0] : 0
dist[0][1] : 3
dist[0][2] : 1
dist[1][0] : 66
dist[1][1] : 59
dist[1][2] : 7
dist[2][0] : 8
dist[2][1] : 2
dist[2][2] : 9

shortest distances between every pair of vertices

```

	0	1	2
0	0	3	1
1	66	59	7
2	8	2	9

Experiment No: 9

Aim: Program to implement 8-Queens' problem using Backtracking.

Program:

```
#include<stdio.h>
```

```
#include<stdlib.h>
```

```
#include<conio.h>
```

```
int t[8] = {-1}; int sol  
= 1;
```

```
void printsol()
```

```
{
```

```
    int i,j; char
```

```
crossboard[8][8];
```

```
for(i=0;i<8;i++)
```

```
{
```

```
    for(j=0;j<8;j++)
```

```
    {
```

```
        crossboard[i][j]='_';
```

```
    }
```

```
}
```

```
for(i=0;i<8;i++)
```

```
{
```

```
    crossboard[i][t[i]]='q';
```

```
}
```

```
for(i=0;i<8;i++)
```

```

        {
            for(j=0;j<8;j++)
            {
                printf("%c ",crossboard[i][j]);
            }
            printf("\n");
        }
    } int empty(int
i)
{
    int j=0;
    while((t[i]!=t[j])&&(abs(t[i]-t[j])!=(i-j))&&j<8)j++;
return i==j?1:0;
}

void queens(int i)
{
    for(t[i] = 0;t[i]<8;t[i]++)
    {
        if(empty(i))
        {
            if(i==7){
printsol();

                printf("\n solution %d\n",sol++);
            }
            else
                queens(i+1);
        }
    }
}

```

```

    }
} int main()
{
    queens(0);
    getch();
    return 0;
}

```

Output:

```

D:\DAA Prac\Queens.exe
Q _ _ _ _ _ _ _ _
_ _ _ _ _ Q _ _
_ _ _ _ _ _ _ Q
_ _ _ _ _ _ Q _
_ _ _ _ _ Q _ _
_ _ _ _ _ _ _ Q
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _

solution 1
Q _ _ _ _ _ _ _
_ _ _ _ _ Q _ _
_ _ _ _ _ _ _ Q
_ _ _ _ _ _ Q _
_ _ _ _ _ Q _ _
_ _ _ _ _ _ _ Q
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _

solution 2
Q _ _ _ _ _ _ _
_ _ _ _ _ Q _ _
_ _ _ _ _ _ _ Q
_ _ _ _ _ _ Q _
_ _ _ _ _ Q _ _
_ _ _ _ _ _ _ Q
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _

solution 3
Q _ _ _ _ _ _ _
_ _ _ _ _ Q _ _
_ _ _ _ _ _ _ Q
_ _ _ _ _ _ Q _
_ _ _ _ _ Q _ _
_ _ _ _ _ _ _ Q
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _

```

Experiment No: 10

Aim: Program to implement 0/1 Knapsack Problem/ Traveling sales person problem using

Dynamic programming **Program:**

```
#include<stdio.h>
```

```
#include<conio.h>
```

```
int ary[10][10],completed[10],n,cost=0;
```

```
void takeInput()
```

```
{ int
```

```
i,j;
```

```
printf("Enter the number of villages: "); scanf("%d",&n);
```

```
printf("\nEnter the Cost Matrix\n");
```

```
for(i=0;i < n;i++)
```

```
{
```

```
printf("\nEnter Elements of Row: %d\n",i+1);
```

```
for( j=0;j < n;j++) scanf("%d",&ary[i][j]);
```

```
completed[i]=0;
```

```
}
```

```
printf("\n\nThe cost list is:");
```

```
for( i=0;i < n;i++)  
{ printf("\n");  
  
for(j=0;j < n;j++) printf("\t%d",ary[i][j]);  
}  
}
```

```
void mincost(int city)  
{ int i,ncity;
```

```
completed[city]=1;
```

```
printf("%d-->",city+1); ncity=least(city);
```

```
if(ncity==999) {  
ncity=0;  
printf("%d",ncity+1);  
cost+=ary[city][ncity];
```

```
return;  
}
```

```
mincost(ncity);  
}
```

```
int least(int c) { int  
i,nc=999; int  
min=999,kmin;
```

```

for(i=0;i < n;i++) {
    if((ary[c][i]!=0)&&(completed[i]==0)) if(ary[c][i]+ary[i][c]
    < min) { min=ary[i][0]+ary[c][i]; kmin=ary[c][i]; nc=i; }
}

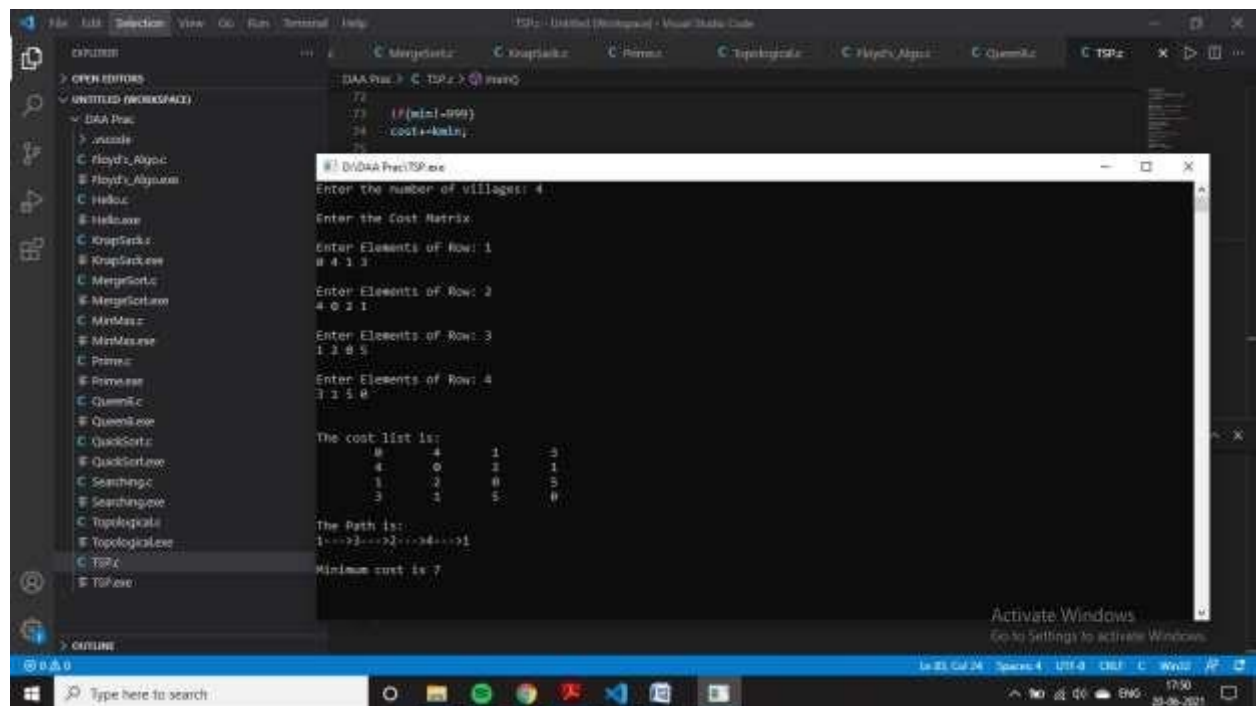
if(min!=999) cost+=kmin;

return nc; }

int main() {
    takeInput();
    printf("\n\nThe Path is:\n"); mincost(0);
    //passing 0 because starting vertex
    printf("\n\nMinimum cost is %d\n ",cost);
    getch(); return 0;
}

```

Ouput:



```

D:\DAA Prac\TSP.exe
Enter the number of villages: 4
Enter the Cost Matrix:
Enter Elements of Row: 1
0 4 1 3
Enter Elements of Row: 2
4 0 2 1
Enter Elements of Row: 3
1 2 0 5
Enter Elements of Row: 4
3 1 5 0

The cost list is:
0 4 1 3
4 0 2 1
1 2 0 5
3 1 5 0

The Path is:
1->2->3->4->1

Minimum cost is 7

```